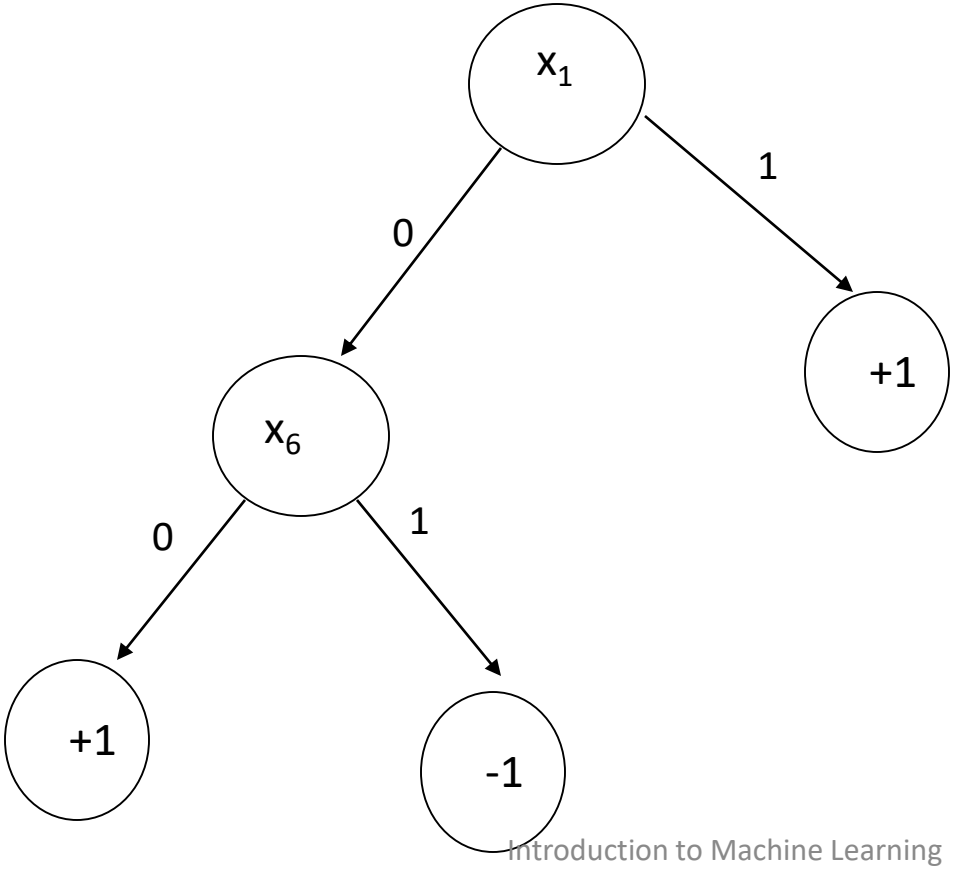
Decision Trees

Decision Trees

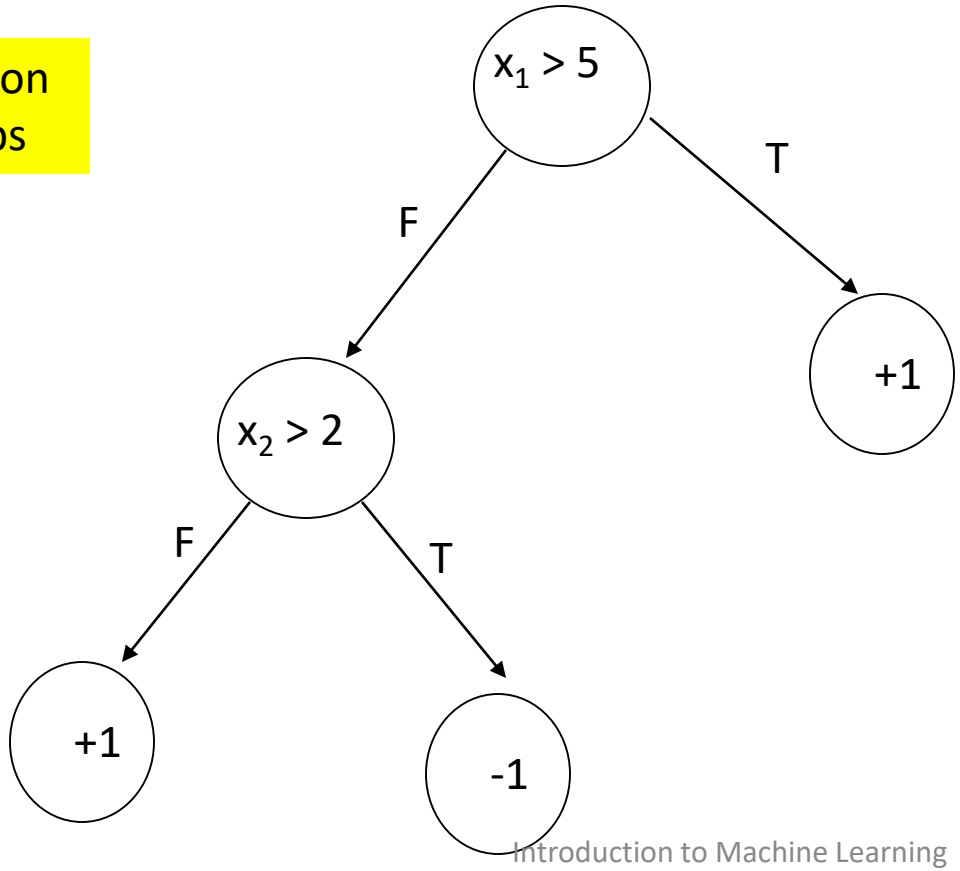
- This week:
  - constructing DT
    - Greedy Algorithm
    - Potential Function
      - upper bounds the error
  - Pruning DT
- Next week:
  - Ensemble methods
    - Random Forest

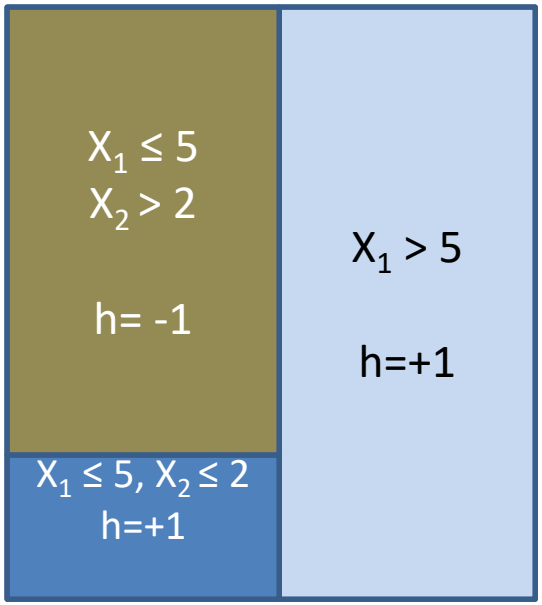
Decision Trees - Boolean



Decision Trees - Continuous

Decision stumps





Decision Trees: Basic Setup

- Basic class of hypotheses H.
  - For example $H=\{x_i\}$ or $H=\{x_i>a\}$
- Input: Sample of examples
  - $S=\{(x,b)\}$
- Output: Decision tree
  - Each internal node from H
  - Each leaf a classification value
- Goal:
  - Small decision tree
    - Good generalization
  - Classifies (almost) all examples correctly.

Decision Tree: Why?

- Human interpretability
- Efficient algorithms:
  - Construction.
  - Classification
- Performance:
  - Reasonable
  - Random Forest
    - State of art

- Software packages:
  - CART
  - C4.5 and C5
  - Many more
    - Even in excel and matlab

Decision Trees: VC dimension

- Vcdim(s,n)
  - Tree size: s leaves
  - Num. Attributes n
- Binary attributes
- Lower bound
  - For any s examples
  - Build a tree with single example per leaf
- Vcdim(s,n)≥s

- Upper bound
  - Number of trees
    - Catalan num.
    - $CN(s)=\frac{1}{s+1}\binom{2s}{s} \approx O\left(4^s/s^{1.5}\right)$
  - Per tree:
    - Attribute per node: n^{s-1}
    - Label per leaf: 2^s
  - Number of functions
    - $CN(s)*2^s n^{s-1} \leq (8n)^s$
- Vcdim(s,n) ≤ s log (8n)

Decision Trees: VC dimension

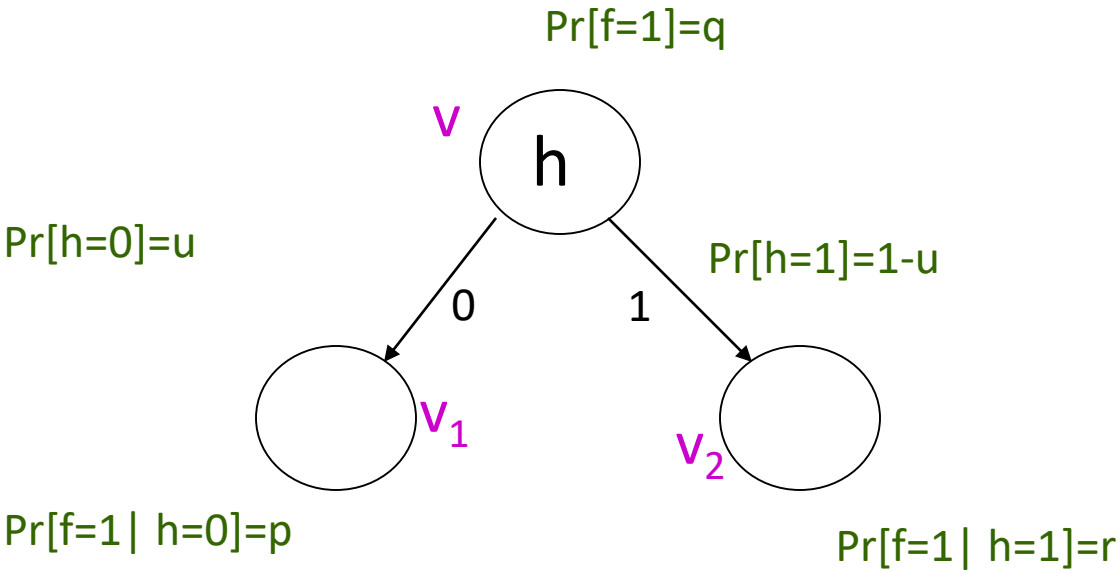
- General attributes
- Each node $h \in H$
  - Vcdim(H)=d
- Methodology
  - Consider m inputs
  - Upper bound number of functions $NF(m)$
  - Shattering m points
    - $2^m \leq NF(m)$
  - Upper bounds m
- Counting trees $\leq 4^s$
- Counting leaves $\leq 2^s$
- Matrix $m \times s$
  - Rows inputs
  - Columns internal split
  - NF(m) bounded by number of matrices
- Counting matrices
  - column has m^d values
  - In total m^{ds}
- $NF(m) \leq 4^s 2^s m^{sd}$
- Vcdim(s,n)=O(sd log(sd))

Decision Trees Algorithm: Outline

- A natural recursive procedure.
- Decide a predicate *h* at the root. 
- Split the data using *h*
- Build right subtree (for *h(x)=1*)
- Build left subtree (for *h(x)=0*)
- Running time
  - Time(m) = O(m) + Time(m^+) + Time(m^-)
    - Tree size <m = sample size
  - Worse case O(m^2) average case O(m log m)

DT: Selecting a Predicate

- Basic setting:



- Clearly: *q=up + (1-u)r*

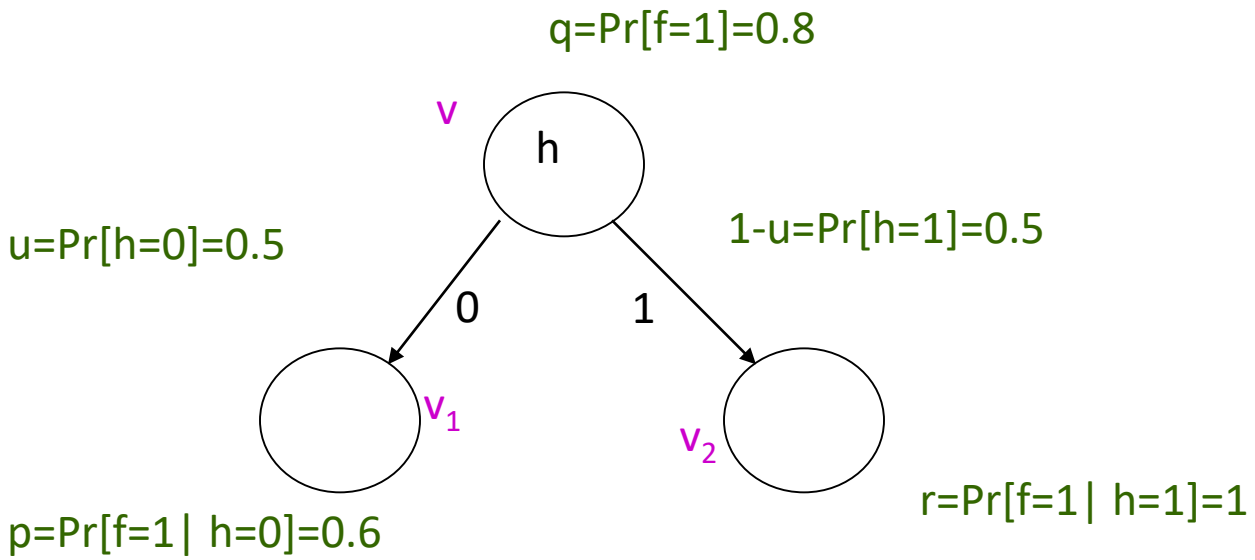
Potential function: setting

- Compare predicates using potential function.
  - Inputs: q, u, p, r
  - Output: value
- Node dependent:
  - For each node v and predicate h assign a value.
    - val(v)= val(u,p,r)
      - Q: what about q ?! What about the probability of reaching v ?!
  - Given a split: $val(v) = u\ val(v_1) + (1-u)\ val(v_2)$
  - For a tree: weighted sum over the leaves.
    - $Val(T) = \sum_{v\ leaf} q_v\ val(v)$

PF: classification error

- Misclassification potential
  - val(v)=min{q,1-q}
    - Classification error.
    - val(T) = fraction of errors using T on sample S
      - In leaves, select the error minimizing label
- Termination:
  - Perfect Classification
    - Val(T)=0
- Dynamics: The potential only drops

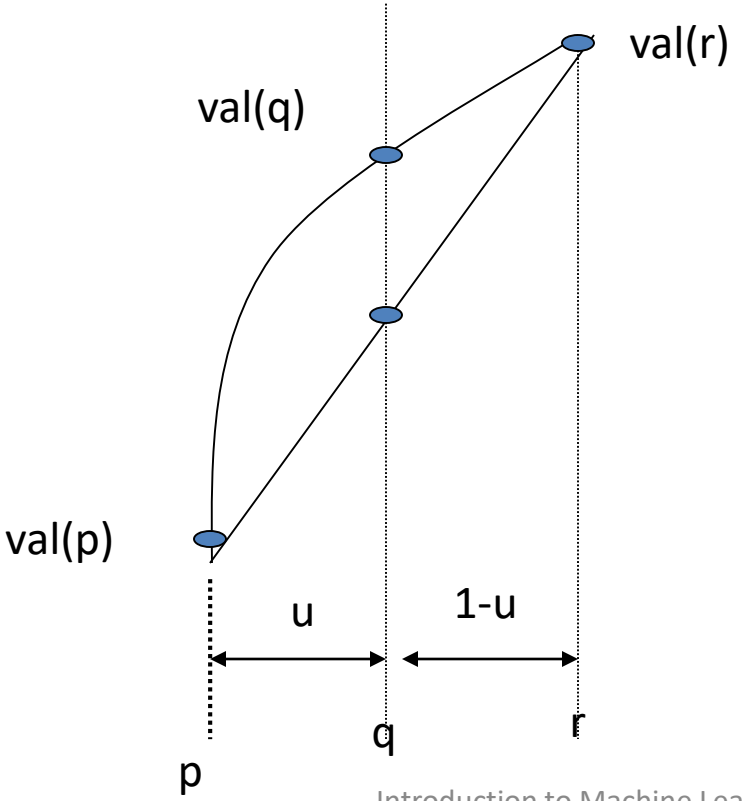
PF: classification error



- Initial error 0.2
- After split 0.5 (0.4) + 0.5(0) = 0.2
- **Is this a good split?**

Potential Function: requirements

- Strictly convex.
- Every change in an improvement
- When zero perfect classification.



Potential Functions: Candidates

- Assumption on val:
  - Symmetric: val(q) = val(1-q)
  - Strictly Convex
  - val(0)=val(1) = 0 and val(1/2) =1/2
- Outcome:
  - Error(T) ≤ val(T)
  - Minimizing val(T) upper bounds the error!

Potential Functions: Candidates

- Potential Functions:
  - val(q) = Gini(q)=2q(1-q) CART
  - val(q)=Etropy(q)= -q log q –(1-q) log (1-q) C4.5
  - val(q) = sqrt{2 q (1-q) } Variance

- Differences:
  - Slightly different behavior
  - Same high level intuition

DT: Construction Algorithm

Procedure DT(S) : S - sample

- If all the examples in S have the classification b
  - Create a leaf with label b and return
- For each h compute val(h,S)
  - $val(h,S) = u_h val(p_h) + (1-u_h)\, val(r_h)$
- Let h' = arg $\min_h$ val(h,S)
- Split S using h' to S^0 and S^1
- Recursively invoke DT(S^0) and DT(S^1)
- Q: What about termination?! What is the running time ?!

Run of the algorithm

- Function val=2q(1-q)
- Basic hypothesis: attrib.
- Initially: val =0.5
- At the root:
  - X_1: (8,5) & (2,0)
    - Val= 0.8*2*(5/8)(3/8)+0.2*0 =0.375
  - X_2: (2,2) & (8,3)
    - Val=0.2*0+0.8*2*3/8*5/8 =0.375

- Example

x	- y	x	- y
11110	- 1	10011	- 0
10010	- 1	10111	- 0
11111	- 1	10011	- 0
10001	- 1	00100	- 0
10101	-1	00000	- 0

Run of the algorithm

- At the root:
  - X_3: (5,3) & (5,2)
    - Val= 0.5*2*3/5*2/5+ 0.5*2*2/5*3/5=0.48
  - X_4: (6,3) & (4,2)
    - Val=0.6*2*0.5*0.5+0.4*2 *0.5*0.5=0.5
  - X_5: (6,3) & (4,2)
    - Val=0.5
- Select x_1
  - Reduction:

0.5- 0.375= 0.125

- Example

x	- y	x	- y
11110	- 1	10011	- 0
10010	- 1	10111	- 0
11111	- 1	10011	- 0
10001	- 1	00100	- 0
10101	-1	00000	- 0

Run of the algorithm

- Root x_1
- Split the sample
- For x_1=0 DONE ! (why?)
- For x_1=1 continue.
  - What about val(x_1)?!
  - For x_2 (2,2) & (6,3) 0.375
  - For x_3 (4,3) & (4,2) 0.4375
  - For x_4 (6,3) & (2,2) 0.375
  - For x_5 (6,3) & (2,2) 0.375
- Select x_2
  - Reduction

  0.375-0.8*0.375=0.015

- Example

x_1 = 1	x_1 = 0
x - y	x - y
11110- 1	00100 - 0
10010- 1	00000 - 0
11111- 1	
10001- 1	
10101 -1	
10011 - 0	
10111 - 0	
10011 - 0	

Run of the algorithm

- Node x_2
- Split the sample
- For x_2=1 DONE !
- For x_2=0 continue.
  - For x_3 (2,1) & (4,2) 0.5
  - For x_4 (3,1) & (3,2) 0.444
  - For x_5 (5,2) & (1,1) 0.4
- Select x_5

- Example

$x_2 = 1$	
x	- y
11110	- 1
11111	- 1

$x_2 = 0$	
x	- y
10010	- 1
10001	- 1
10101	-1
10011	- 0
10111	– 0
10011	- 0

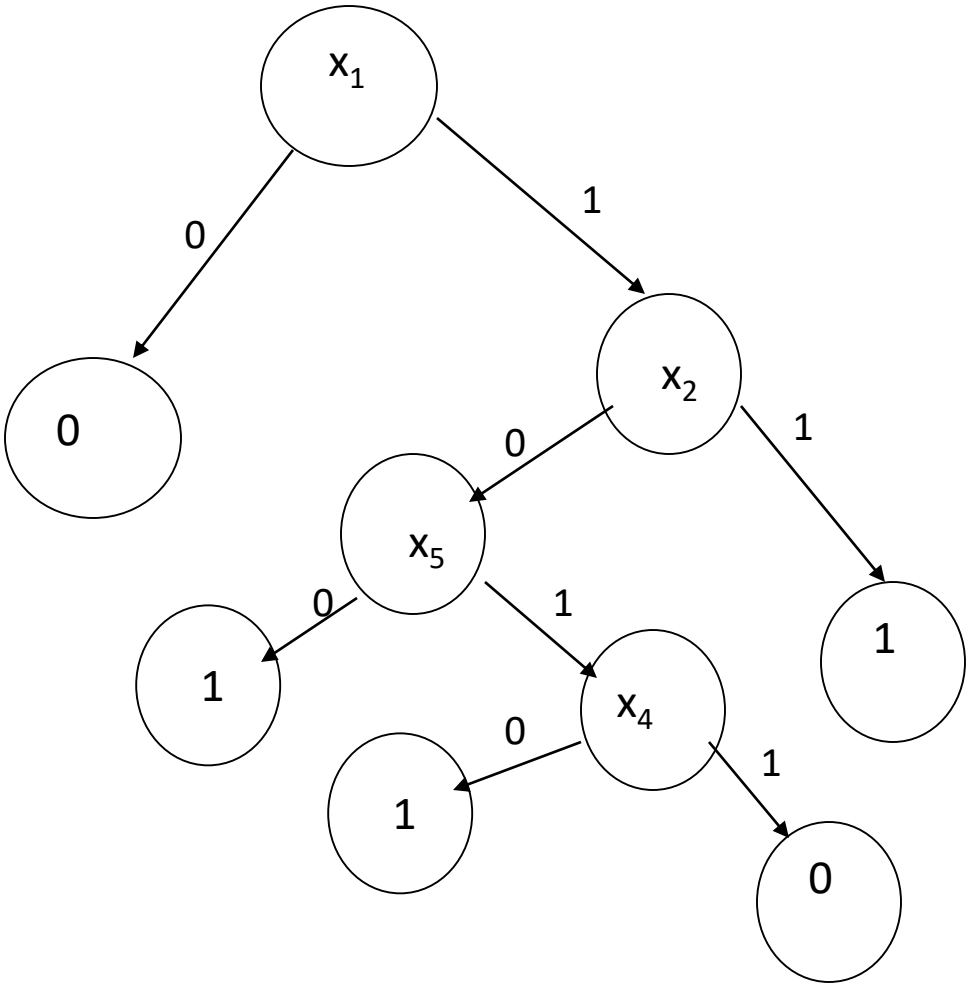
Run of the algorithm

- Node x_5
- Split the sample
- For x_5=0 DONE !
- For x_5=1 continue.
  - For x_3 (2,1) & (3,1) 0.266
  - For x_4 (3,0) & (2,2) 0
- Select x_4 DONE !!

- Example

$x_5 = 1$		$x_5 = 0$	
x	- y	x	- y
10001	- 1	10010	- 1
10101	-1		
10011	- 0		
10111	– 0		
10011	- 0		

Resulting tree



DT: Performance

- DT size guarantee
  - Greedy does not have a DT size guarantee
    - Consider $f(x) = x_1 + x_2 \mod 2$ with d attributes
  - Computing the smallest DT is NP-hard
- Boosting Analysis:
  - Next week
    - Assume a weak learner ($1/2 + \gamma$)
  - Bound DT size
    - $\exp\{O(1/\gamma^2\ 1/\varepsilon^2\ \log^2 1/\varepsilon)\}$ Gini/CART
    - $\exp\{O(1/\gamma^2\ \log^2 1/\varepsilon)\}$ Entropy/C4.5
    - $\exp\{O(1/\gamma^2\ \log 1/\varepsilon)\}$ Variance

Decision Tree Pruning

Problem Statement

- We like to output small decision tree
  - Model Selection
- The building is done until zero training error
- Option I : Stop Early
  - Small decrease in index function
  - Cons: may miss structure
- Option 2: Prune after building.

Pruning

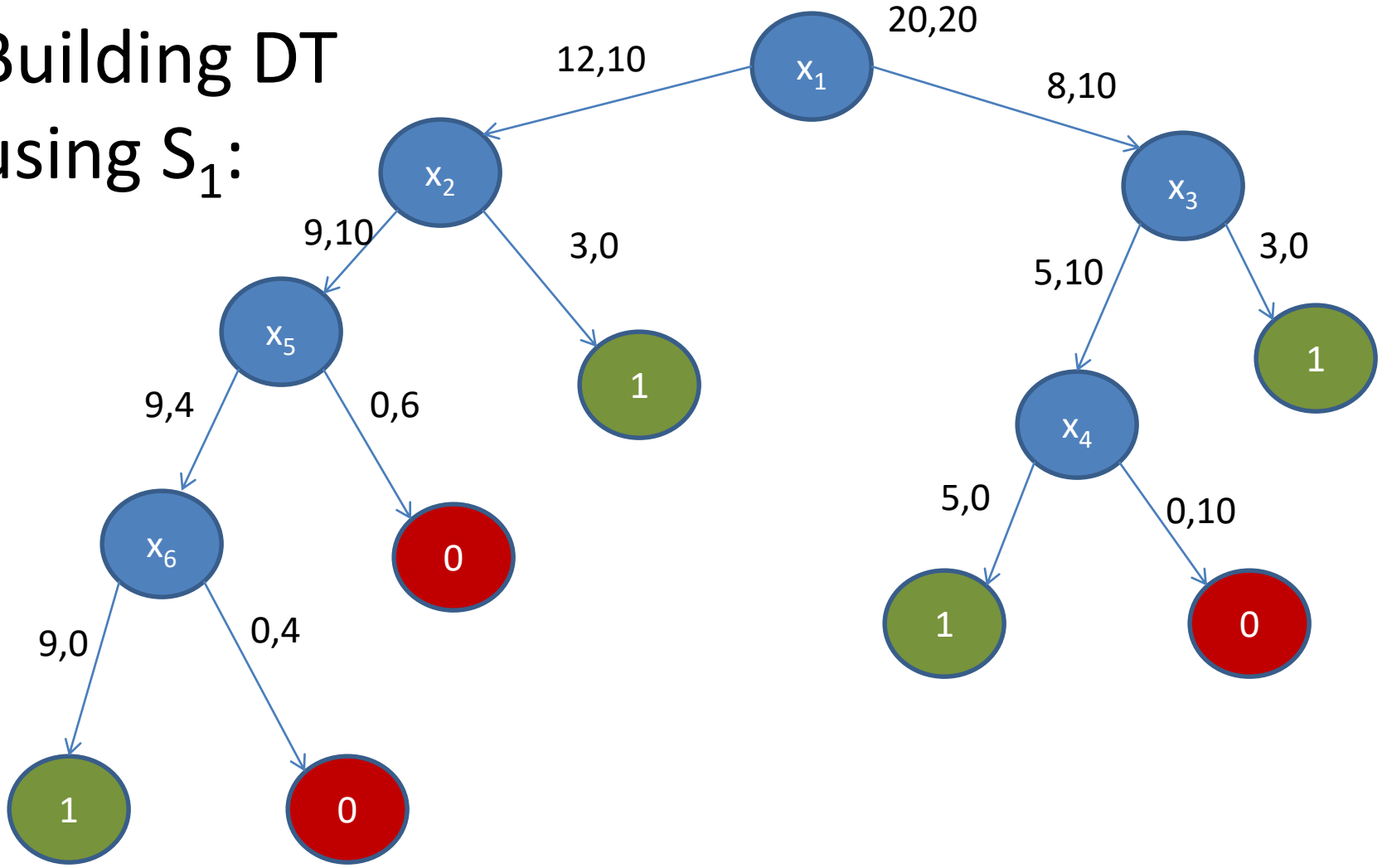
- Input: tree *T*
- Sample: *S*
- Output: Tree *T'*
- Basic Pruning: *T'* is a sub-tree of *T*
  - Can only replace an inner node by a leaf
- More advanced:
  - Replace an inner node by one of its children

Reduced Error Pruning

- Split the sample to two part S_1 and S_2
  - Use S_1 to build a tree.
  - Use S_2 to decide when to prune.
- Process every inner node *v*
  - After all its children have been processed
  - Compute the observed error of T_v and possible *leaf(v)*
  - If *leaf(v)* has less errors replace T_v by *leaf(v)*
- *Alternative: require the difference to be statistically significant*
  - *Can be theoretically analyzed*

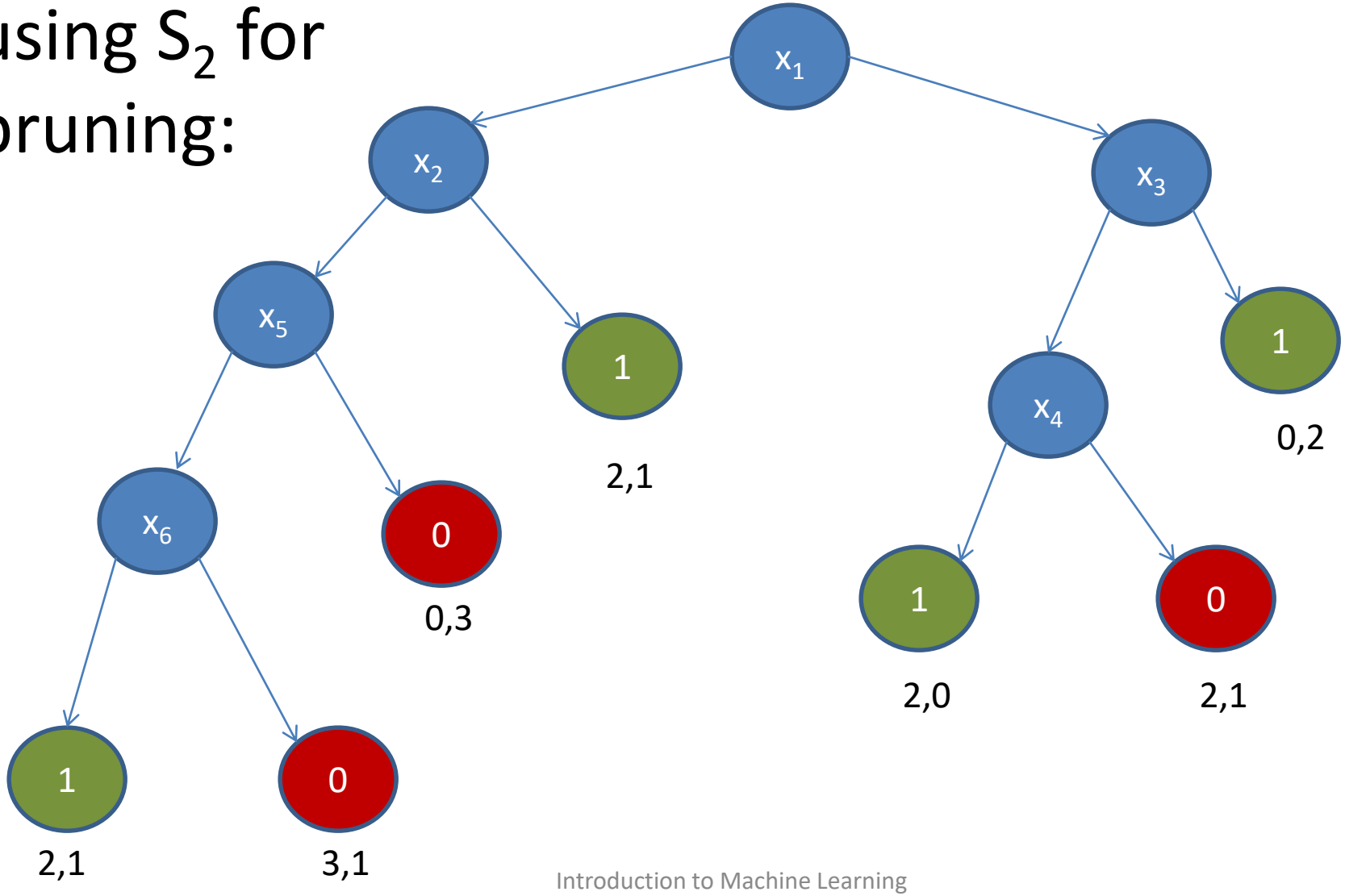
Reduced Error Pruning: Example

Building DT using S_1:

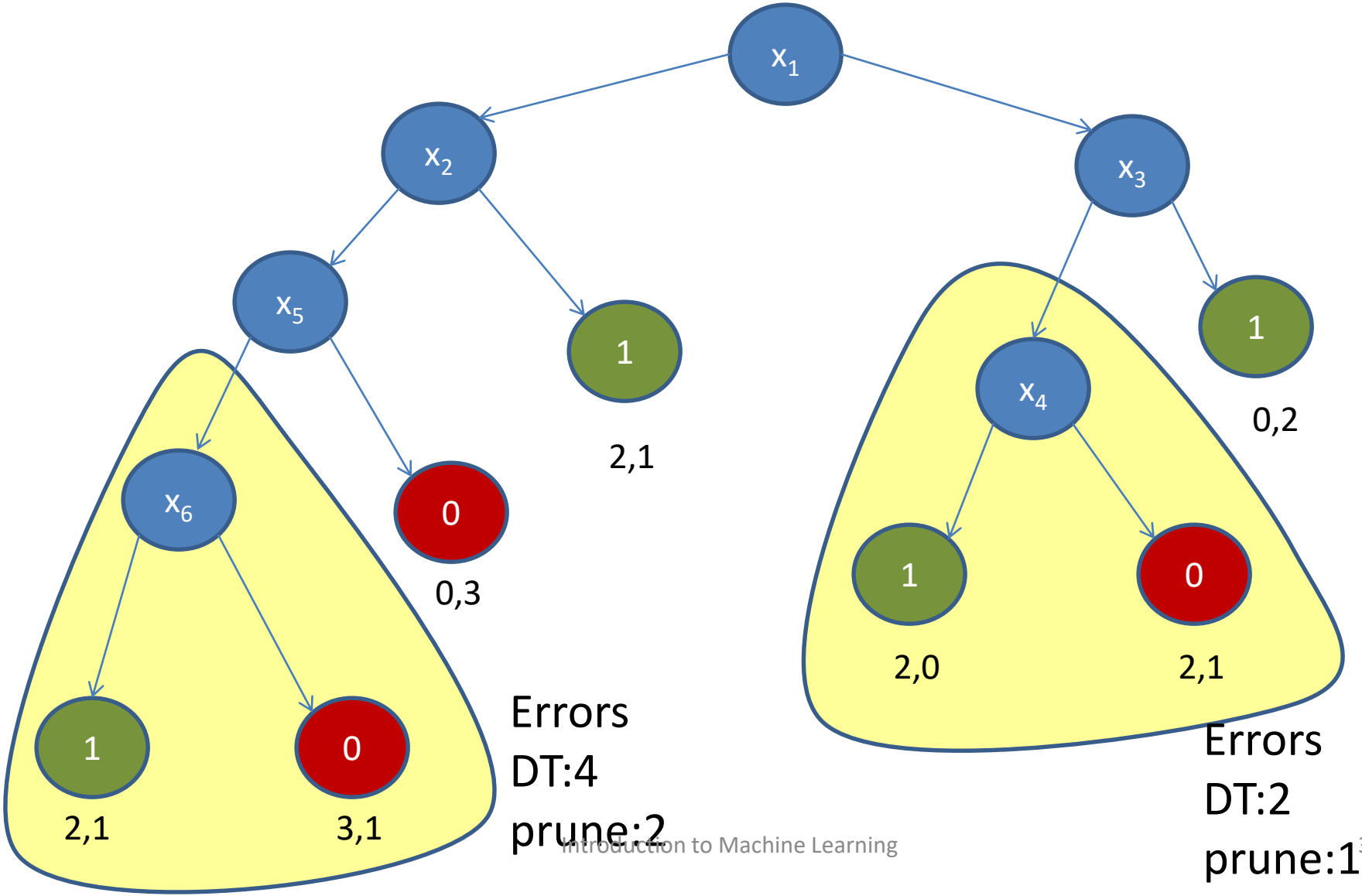


Reduced Error Pruning: Example

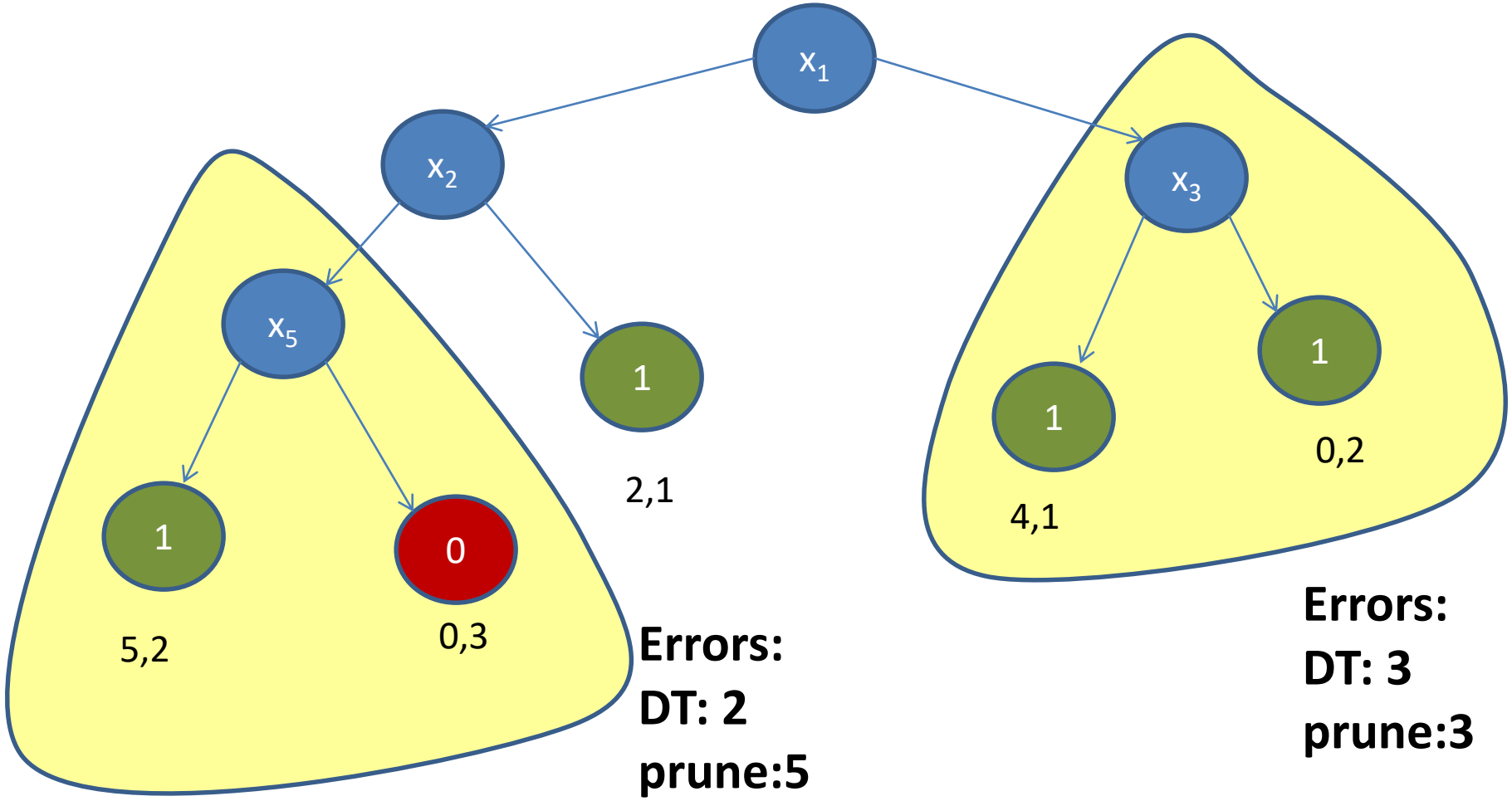
using S_2 for pruning:



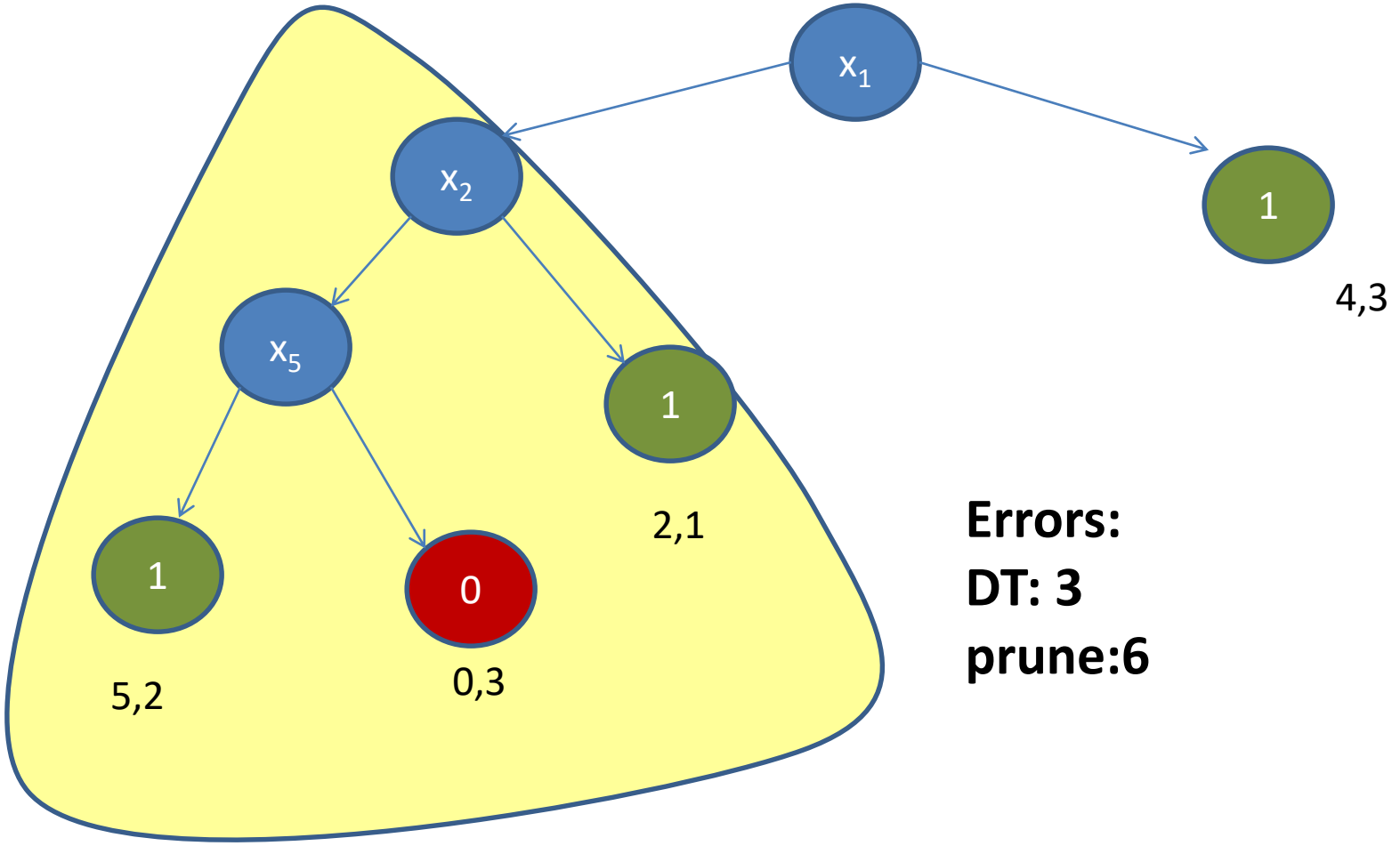
Reduced Error Pruning: Example



Reduced Error Pruning: Example

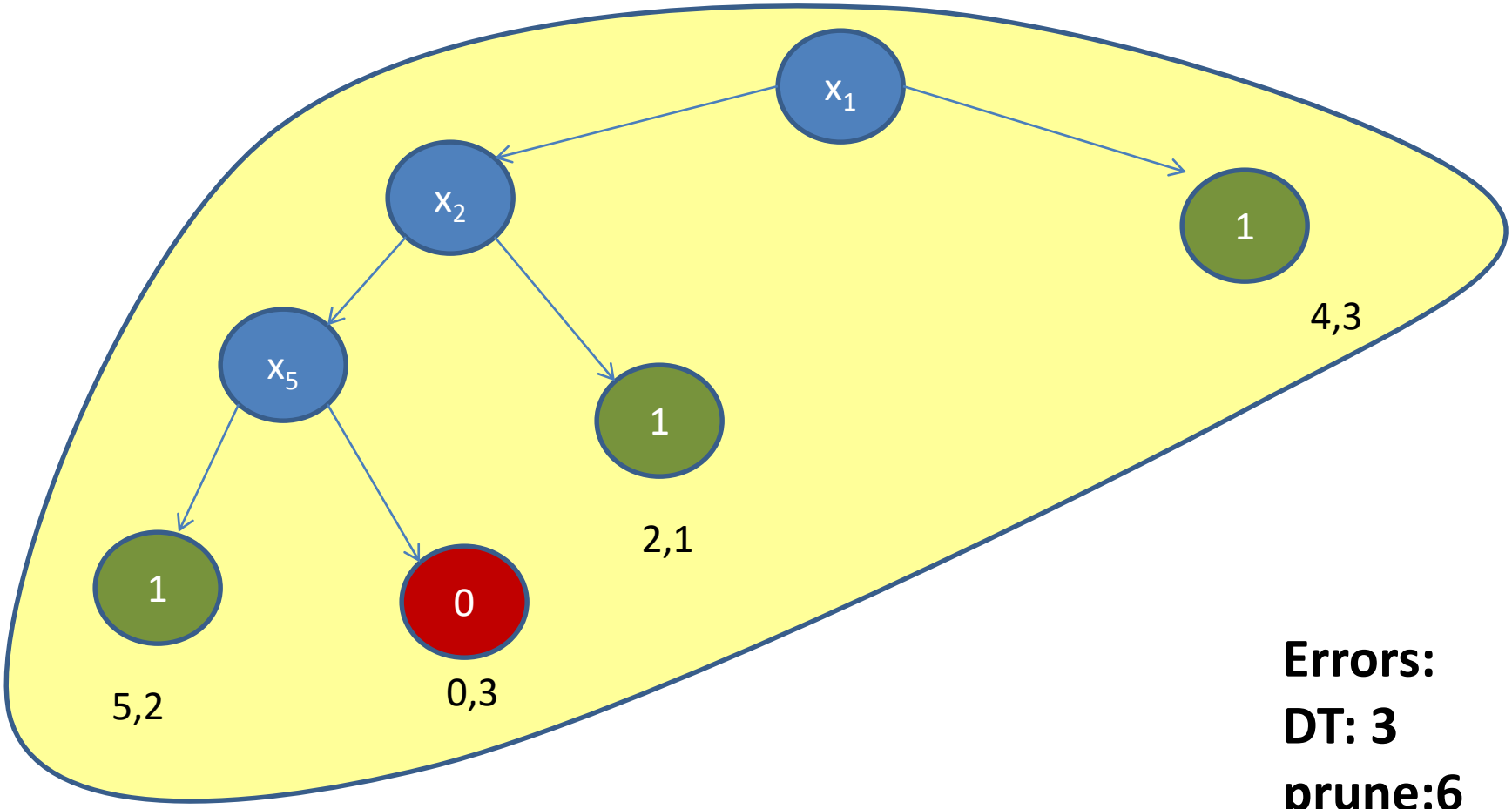


Reduced Error Pruning: Example



Errors:
DT: 3
prune:6

Reduced Error Pruning: Example



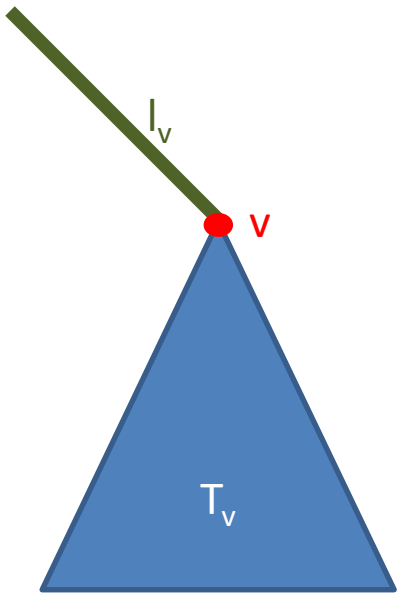
Errors:
DT: 3
prune:6

Bottom-up pruning

- RED is also bottom up
  - Using held out set
- We can do the pruning using confidence intervals
  - SRM
- High level:
  - Prune if leaf is not much worse than the subtree

Bottom-up pruning: SRM

- Parameters



  - m_v = sample at v

- Conservative criteria α
  - With probability 1-δ, if
    $\widehat{\varepsilon_v}(T_v) + \alpha(m_v, |T_v|, l_v, \delta) \geq \widehat{\varepsilon_v}(\emptyset)$
  - Then
    $\varepsilon_v(T_v) \geq \varepsilon_v(\emptyset)$
- Each non-pruning justified!
- Example: Boolean attrib.
- $\alpha = \sqrt{\frac{(l_v + |T_v|)\log(n) + \log(\frac{1}{\delta})}{m_v}}$

Bottom-up pruning: SRM

- Given T:
  - T_{opt} the optimal pruning
    - Minimizes the error
  - T_{srm} our pruning

- Lemma: with prob 1-δ

T_{srm} is a subtree of T_{opt}

- Follows from the conservativeness.

- Theorem:

$$err(T_{srm}) - err(T_{opt}) \leq \sqrt{\frac{|T_{opt}|}{m}\beta}$$

Where

$$\beta = O\left(\log\frac{|T_{opt}|}{\delta} + h_{opt}\sqrt{\log n + \log\frac{m}{\delta}}\right)$$

Pruning: Model Selection

- Generate DT for each pruning size
  - compute the minimal error pruning
  - At most *m* different decision-trees
- Select between different pruning:
  - Hypothesis Validation
  - Structural Risk Minimization
    - Any other index method

Finding the minimum pruning

- Procedure Compute
- Inputs:
  - k : number of errors
  - T : tree
  - S : sample
- Output:
  - P* : pruned tree
  - Size* : size of P

- Compute(k,T,S,P*,size*)
- IF *IsLeaf(T)= TRUE*
  - IF *Errors(T)* $\leq k$
    - THEN *size*=1*
    - ELSE *size** $= \infty$
  - *P*=T*; return;
- IF *Errors(root(T))* $\leq k$
  - *Size*=1*; *P*=root(T)*; return;

Procedure compute

- For i = 0 to k DO
  - Call Compute(i, T[0], S_0, $P_{i,0}$, $size_{i,0}$)
  - Call Compute(k-i, T[1], S_1, $P_{i,1}$, $size_{i,1}$)
- Size* = minimum $\{size_{i,0} + size_{i,1} + 1\}$
- i* = arg min $\{size_{i,0} + size_{i,1} + 1\}$
- P* = MakeTree(root(T),$P_{i^*,0}$, $P_{i^*,1}\}$
- Return
- What is the time complexity?

Hypothesis Validation

- Split the sample S_1 and S_2
- Build a tree using S_1
- Compute the candidate prunings
  - $P_1, \ldots, P_m$
- Select using S_2
  - $T^* = Arg\ min\ error(P_i, S_2)$
- Output the tree T*
  - Has the smallest error on S_2

SRM

- Build a Tree *T* using *S*
- Compute the candidate prunings
  - $P_1, \ldots, P_m$
  - k_d the size of the pruning with *d* errors
- Select using the **SRM** formula

$$\min_d \{error(S, T_d) + \sqrt{\frac{k_d}{m}}\}$$

Drawbacks

- Running time
  - Since $|T| = O(m)$
  - Running time $O(m^2)$
  - Many passes over the data
- Significant drawback for large data sets

More on Pruning

- Considered only leaf replacement
  - Substitute a sub-tree by a leaf
- Other popular alternatives
  - Replace a node by one of its children.
- Reduce error pruning
  - Conceptually similar
- Model selection

Decision Trees

- This week:
  - constructing DT
    - Greedy Algorithm
    - Potential Function
      - upper bounds the error
  - Pruning DT
- Next week:
  - Ensemble methods
    - Random Forest